BAIT SOFTWARE

The present invention relates to software piracy. More specifically, the present invention relates to a method for the restriction or prevention of software piracy.

BACKGROUND OF THE INVENTION

The problem of software piracy arises from the nature of software itself, in that any copy of software is a binary copy, a perfect copy that will run the same as the original. In fact, the only thing distinguishing two binary copies of any software is their respective locations in time and space. Therefore the relapse of one unprotected copy of any program opens up the possibility of an infinite number of copies being made.

Any user can infringe the copyright of almost any software without any chance of being noticed or caught as long as his machine is not physically examined for pirated software.

Software piracy represents an enormous revenue loss for every software developer, which loss can be a multiple of the actual earned revenue of that developer. This invention enables the software developer to restrict or prevent software piracy.

Several devices and methods have been introduced over time to limit or constrain software piracy but they all fail in some important and logical manner to absolutely prevent piracy. Piracy must be prevented in an absolute manner since the release out of the hands of the developer of even one unprotected copy, permits endless copying and distribution by hackers, infringers and pirates.

There are many TrialWare products (such as TimeLock) which allow an application developer to distribute a demo or trial version of his software which "expires" on the user's computer after a set number of accesses or a set period of time. The problem occurs after the user has paid a fee and "unlocked" the software so that it becomes fully functional.

At that point, the software can be distributed to infringers by copying the installed program. There may be changes required to the setup or Registry of the infringers computer to incorporate certain setup or enabling features but a semi-skilled hacker can accomplish this. In fact, the Warez sites on the Internet are places where unprotected copies of many products can be downloaded. The "Hacking" of the protection features of TrialWare products is a game for certain people and status in that forum is gained by the number and complexity of protected programs which have been hacked and re-released as "freeware".

Baitware works by including multiple levels of protection.

A At the front end, Baitware employs trialware features (limiting the initial time or the number of initial accesses - to be set by the developer) to encourage the user to register the installation of the application.

B Upon registration, the RID (Registration Identifier) server can use the MIV (Machine Identifier Value) of the user to calculate an unlock code which will only unlock that single copy of the application as already installed on the valid (paid) users system.

C Upon registration, the RID server now has the contact information required to ensure that only one copy of each valid and paid RID is installed, preventing multiple installation of a single copy

D The RID server now has a complete list of each valid user, to whom will be sent, from time to time, special codes to unlock or deactivate the buried features, which are the fallback defense of Baitware should the front end defenses ever fail.

These fallback defenses are not known to the users in advance and may include multiple, cascading drop dead dates which freeze the application further use and invite the infringer to contact customer service for registration. Drastic action such as overwriting portions of the EXE in some random fashion which would totally disable further use, are also possible.

E If desired, the RID server can be set to receive information sent at discrete intervals by any installed application, whether valid or not.

The transmission of this information would be invisible to the user and would inform the RID server that infringement has taken place, and give the email address of the infringer.

What this application discloses and teaches is a safe method for the developer to supply an antidote to the legal users while preventing infringers from taking that antidote under any circumstances.

It is an object of the invention to overcome disadvantages of the prior art.

The above object is met by the combination of features of the main claim, the sub-claims disclose further advantageous embodiments of the invention.

SUMMARY OF THE INVENTION

The present invention relates to the restriction or prevention of software piracy.

According to the present invention there is provided a method and means of preventing software piracy comprising the steps of:

a.)Treating the software to use the Burnin process, wherein Burnin performs the initialization steps of:

i.)Calculating a trial period ($T_{(x)}$) from first execution, after which the program becomes expired, whereby expired programs are disabled via Burnin depending on software developer preference,

ii.)Assigning an undisclosed "absolute expiry date" or death-date, $D_{(d)}$, set for a date which is given by:

$D_{(d)} = D_{(x)} + T_{(d)}$, where $T_{(d)}$ is a time period of at least 3 times the time period $T_{(u)}$

iii.)Dynamically generating an MIV value using the current computer software and hardware configurations,

iv.)Prompting the user for user information and the developer supplied RID, where the set of user information is specified by the software developer,

v.)Recording all data from steps i-iv in random and developer-specified locations in the executable file for the software,

vi.)Recording all data from steps i-iv in random and developer-specified locations within the current system-configuration-definition or registry,

b.)Distributing the BurnIned software in version $V_{(x)}$ as TrialWare on the date $D_{(x)}$ via any and all distribution channels, these channels including:

i.)Floppies

ii.) CD-ROM

iii.) Internet

iv.) Intranet

v.)Extranet

c.)Generating and dispensing a new and unique RID value for each copy of software sold to a user, the RID value being unique across all versions of the software throughout its life-time,

d.)Augmenting the Burnin process with a new step to re-cover user-registration data for paid users, including the MIV and RID values, from all installations and/or first-executions of the software, and to further store this data in the Customer Database,

e.)Constructing a free upgrade of the software in version $V_{(x+1)}$ which includes the Customer Database constructed in step d; version $V_{(x+1)}$ being able to upgrade legal and illegal copies of the software in version $V_{(x)}$ via the initialization steps of:

i.)Searching the Customer Database for a matching MIV-RID pair,

ii.)If found, resetting the absolute expiry date, $D_{(x)}$, to Y years from current date, where Y is defined as the life-time of the software in version $V_{(x-1)}$.

iii.)If not found, recording all registration data and dynamically generated data to the central data base of step d, identifying a particular MIV as an illegal, but still upgraded user.

f.)Distributing the software in version $V_{(x-1)}$ constructed in step-e as a free upgrade within a time period, $T_{(u)}$, after the release date, $D_{(x)}$, where $T_{(u)}$ is specified by the software developer as the time period required for the product to reach 100% of currently legal users.

g.)Disabling each copy of the software in version $V_{(x-1)}$ on the death date $D_{(d)}$; this process, which is activated the next time the software is run on or after the death date, comprising the steps of:

i.)Executing and/or undertaking all additional actions specified by the software developer.

ii.)Further disabling the software in version $V_{(x)}$, thereby also disabling any future re-installations of the software in version $V_{(x)}$.

h.)Contacting all illegal users recorded in the Customer Database on the death date $D_{(d)}$, and communicating all software developer specified information.

þ

This summary of the invention does not necessarily describe all necessary features of the invention but that the invention may also reside in a sub-combination of the described features.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the invention will become more apparent from the following description in which reference is made to the appended drawings wherein:

FIGURE1 shows an embodiment of ann aspect of the present invention. This figure shows a computer 100, and the Baitware software 104, acting on any number of software applications 102, within the computer.

FIGURE 2 shows an aspect of th present invention indicating the basic components of the system described hererin.

FIGURE 3 shows another aspect of the present invention, and indicates the Lockdef record.

FIGURE 4 shows another aspect of the present invention, and indicates the Program Control Block (PCB), Registry Control Record (RCR), Time Lock Type (TLT) constants for LockDef.type, and Time Lock Action (TLA) constants for LockDef.action.

FIGURE 5 shows another aspect of the present invention, and indicates the data elements duplicated in the PCB and RCR.

FIGURE 6 shows another aspect of the present invention, and indicates the assocaitions of each program file in the system of the present invention.

FIGURE 7 shows another aspect of the present invention, and indicates the rquired Baitware development.

FIGURE 8 shows another aspect of the present invention, and indicates the setup of the Burlin module.

FIGURE 9 shows another aspect of the present invention, and indicates the functions carried out by Burnlin.

FIGURE 10 shows another aspect of the present invention, and indicates fall back defenses as described hererin.

FIGURE 11 shows another aspect of the present invention, and indicates the errors which may occur in the Baitware system.

FIGURE 12 shows another aspect of the present invention, and indicates the steps for Baitware operation with softwarre downloaded from the Internet.

DESCRIPTION OF PREFERRED EMBODIMENT

The present invention relates to software piracy. More specifically, the present invention relates to a method for the restriction or prevention of software piracy.

The following description is of a preferred embodiment by way of example only and without limitation to the combination of features necessary for carrying the invention into effect.

Components & Concepts

The Baitware method.

The first step is to bind the application directly to the machine. Using a product such as BurnIn, the first step is to mate a software program irrevocably to a specific machine. under the BurnIn system, the user inputs certain personal data as well as corporate data into a form at the first install. This information together certain configuration of the users machine creates a Machine Identifier Value (MIV).

BurnIn furthers the process by "branding" this information directly into the executable of the program at the first installation. Once the information which has been burned into the program is encrypted and randomly placed in the .EXE, the end user or any hacker should not be able to find, let alone alter the inserted information. There are alternative for providing machine information. For

example, every Intel CPU has a unique identifier that can be used at the first install to create the unique reference point.

At this point, at every activation of the program, the program will run out and test that the machine on which it is installed is the one which has the unique (and not user definable) identifier. If the response is correct, the application will run, if not, then not.

Using commonly available devices (such as telephone, email, fax, on-line) the end user is prompted to return the new MIV information together with the rest of the information in the form to the application developer. The developer may require this information as part of the registration process. To increase the difficulty of decrypting the essential Baitware information, blocks of meaningless data can also be included at random locations in the EXE to further increase the difficulty.

The specific copy of the application which has been installed is now fixed to a known system, The RID server will prevent further installation of that same identified copy to another machine. Should the user break through the defenses that prevent the copying of an already installed application or the installation of a copy without final authorization from the RID server, the fall back defenses will be required to come into play.

At some point, the developer releases an upgrade. This can be either on-line (freely downloadable) or on a CD-ROM. On that CD-ROM or download will be a list of Legal Users Names and Legal MIVs. The CD-ROM will check the actual MIV of the machine and regardless of whether the application being upgraded is legal or not, it will accomplish the upgrades without disruption. But only in the case of a legal copy (which is verified at upgrade time by searching for and verifying the MIV) will the fall back defences such as a secret drop-dead date be disabled.

This method causes every user (legal and otherwise) to swallow and ingest an application that has been baitwared, in effect, the user has swallowed a time bomb, which he cannot discover and cannot eliminate. Only the developer can eliminate or nullify the fallback defenses, by offer the antidote to the Baitware poison.

A crucial feature of this system is the delay before the fall back defenses activate. This time is developer specified and will depend on his preferences and knowledge of his own customer base. The central feature of this delay is that potential infringers will create data which is formatted for that particular application. A developer may wish to wait a long time before allowing the fall back defences to activate, thereby trapping infringer files and data which cannot be used without registering and paying for the application. IN certain cases, no such data is created and the developer may prefer a much shorter time before the fall back defences activate.

Overview

Figure 1 is a diagram showing a computer 100, and the Baitware software 104, acting on any number of software applications 102, within the computer.

Figure 2 shows the basic components of the system. A BaitwareLib.DLL file 200 is a compiled run time library containing a PCB, an RCR, and an instance of the Baitware class which contains Baitware instructions. An application executable file 202 contains any applications code, as well as a PCB instance. The computer registry 204 contains up to three instances of the RCR.

LockDef Record

The Lockdef record, shown in Figure 3, contains run time information, together with persistent data (preserved from one execution to the next). The data types shown as given in Microsoft MFC/C++. The top portion gives the set of input data, required during installation of an application. The next portions can be viewed as input or output, depending on the process involved. Lockdef forms the basis for most other records used in Baitware.

Program & Registry Control

With reference to Figure 4, the Program Control Block (PCB) 400 contains the set of data elements indicated in Figure 3. The Registry Control Record (RCR) 402 contains the indicated elements from LockDef 300, as well as an offset into the starting location of a PCB in the application executable file 202.

Constants

The different types for a LockDef occurrence are each represented by a different constant 404. Similarly, the different types of LockDef actions are also each represented by a different constant 406. Note that these constants are mutually inclusive and are therefore implemented as bit-flag values.

Member Duplication & Purpose

Figure.5 shows a table showing which data elements are duplicated in both the PCB and the RCR. This table also shows the purpose of each data member in each of the PCB and the RCR. The following notes apply to the superscripted numbers in Figure.5

(1) the RCR also contains additional members as shown in Figure.1

(2) hid is only required when the TLA_HELP action flag is supplied.

(3) url is only required when the TLA_CONNECT action flag is supplied.

Program File, Registry, and Record Associations

Figure 6 is a table showing the associations of each program file in the Baitware system with registry keys as well as with the PCB and RCR records. "<app><ver>.exe" is a self expanding archive with Baitware supporting code which immediately calls "setup.exe" after expansion is complete. "setup.exe" is a developer-generated setup program with Baitware supporting code which initially deletes the "<app><ver>.exe" file, then after installation is complete, it immediately activates the "app.exe" executable file. "app.exe" is the software

application being protected by Baitware which contains Baitware supporting code. The first action of "app.exe" is to delete the file "setup.exe".þ

Common Files for a Baitware Library

Figure 7 shows the files required for Baitware development. BaitwareLib.DLL 200 is a run-time library as described above; it must be located in a directory area such that the OS can find it (e.g. as part of a search path). BaitwareLib.LIB 700 is linked with an application to provide that application with Baitware supporting code. BaitwareLib.h 702 is a header file; this header must be included in in the application which wants to use BaitwareLib classes and functions. HexBuffer.h 704 is a header file; this header contains the declaration of the hex-key-string which is later burned-in with the PCB. It can be included in one and only one file of the application.þ

BurnIn

The "BurnIn" portion of the Baitware process is a developer tool kit which provides partial piracy protection for any application using a combination of the following information:

1) an Expiry time period and/or maximum number of executions.

2) a Machine Indentifier Value (MIV) which is automatically generated from the computer's characteristics.

3) User information; this is prompted for by the Burnin process according to developer-defined parameters.

All this information is saved (in encrypted form) in the following storage areas:

EXE-file. This is the .EXE file for the application. The data is written in developer defined locations, where each datum may be written at a different location. These locations provide immediate access to encrypted data buffers to any procedure in the application.

Burnin-file. This is a Burnin-definition file stored in the directory of the application. It contains encrypted data written at developer-defined locations. The rest of the file is filled in with random data.

Registry. The encrypted data is stored in the current system registry as data-values for developer-defined registry keys (or key-paths).

Using this information in the various locations, the BurnIn process can ensure the validity of the program being executed by the current user on the current computer.

The setup of the BurnIn module inside the application requires that certain steps be taken by which the application developer inserts and integrates the BurnIn module into his application (see Figure 8). Code is created 800 to test for the presence of the Baitware.DLL. Code is created 802 to ensure the HexBuffer.h file is present in at least one .CPP file in the application to be protected. Code is created

802 to properly link and align the applications project settings with the BaitwareLib.LIB file. The BaitwareLib.h file must be included 806 in all .CPP files. A call 808 to the BaitwareInit.h() must be inserted in the CXapp::Initinstance(). Code must be written 810 to create the BaitwareInit(), which module has the components shown in 812.

Once installed on the users system, BurnIn carries out a series of functions at every time when the application is run (Figure 9). These functions verify that the application is being run only on a valid system. Any and all errors result in the termination of the application. At the Start, verification is sought that the RegKey exists in the registry 900. If the HexKey is found, then the registry data is loaded and stored 902 in the RCR. Then the PCB offset in the .EXE file 904 is fetched the RCR. This permits the loading of the PCB data from the .EXE 706. If the RegKey is not found in 900, then an attempt is made to find the PCB in the .EXE file 908. If the HexKey is not found 910, then Error 02 error results 916 and the application terminates. If the HexKey is found 910, then the process proceeds to load the PCB data from the .EXE file and create the PCB 906. The process tests the PCB for validity 912 under two paths A/B. If not valid in either case the result is Error01 914 and the application terminates. Under path A, IsBurned() is tested 918 . If unsuccessful, the PCB and RCR are set 930 from the LockDef object values. The PCB is burned into the .EXE files 932. The RegKey is then created in the registry 934 and the values from the RCR are added. The process then tests to see if the RCR matches the PCB 926. If when the process tests for the IsBurned()

918, a return of YES results in Error03 920 . Under path B, the process again tests for IsBurned() 922. If unsuccessful, Error04 results 924 and the application terminates. If successful, the process matches the RCR with the PCB 926. A result of no gives rise to Error05 928 and the application terminates. A YES result brings the process to the Active Create module 936. A successful result leads to IsExpired() 938 and the End of the process. An unsuccessful result in Active Create also brings us to the end.

At the time of BurnIn, the fall back defenses can be installed in the .EXE using one or more drop dead dates. These dates are checked from time by the installed application and if reached would cause the application to fail loading (Figure 10). At the start, the burned application is initialized 1000. Failure to initialize gives rise to Error06 1002 and the application terminates. After successful initialization, the expiry date is checked 1004. If not expired, the process returns false 1006 and continues to run. If the application has expired then the RegTheUser module is activated 1008 to carry out a predetermined set of actions, which were specified by the developer. If the actions specified permits the application to continue, this is then the case 1010. If the specified actions are to terminate or to connect the user to the RIDSERVER, then the application terminates.

Baitware Errors

Figure 11 is a diagram showing the errors which are most likely to occur in the Baitware system. These error conditions 1100 are referenced throughout the

various process diagrams. Each error also has a probable cause 1102; the cause for any one error cannot be determined absolutely. The table 1102 lists the most probable causes.þ

Download to First Execution

Baitware works equally well for software downloaded from the Internet. Figure 12 shows the steps in such a process. First, the user downloads 1200 an executable archive from the Internet. Next, the user executes the downloaded file 1202 to expand and retrieve the files therein. This process involves the creation and validation of a Baitware object 1204, as well as the expansion of all files 1206 in that archive. The user has the option of canceling this process at any time 1222; however, at this point a user cancellation is too late since the Baitware validation process has already occurred. Next, the setup program (extracted from the archive) is automatically executed 1208. Again, this process involves the creation and validation of a Baitware object 1210, but this time the object is created and validated for the setup program. Once validated, the normal installation steps 1212 are undertaken. Again, the user may terminate the process at any time 1222. Upon the completion of the installation a dialog is displayed 1214, informing the user that installation is complete. Once the user presses the OK or CONTINUE button on this dialog, the software application is automatically executed 1216. Now a Baitware object is created and validated for the software application 1218. Finally, the software application can start its normal processing 1220; at this point, the application has been successfully encoded and validated via the Baitware process.þ

Registration-Identifier (RID)

Burnin presently incorporates user entered data (name and address) plus machine characteristics. The form which Burnin prompts a user is expanded to include a unique Registration Identifier (RID) which will be branded into the product together with the other information. The RID and the MIV can then be used as keys into the developer's customer database, to bind a RID (i.e. a copy of the product) directly to a user (an MIV).

The RID can be supplied to the customer in any way, including: adhesive label, printed sheet, and over the phone.

The RID is generated by the developer, starting at a base number (e.g. 0) and incrementally dispensed. A RID value is never re-used throughout the life-time of the application; it must remain unique across all copies of all versions of the application.

The RID Server

This solution envisions adding the RIDServer module to BurnIn. The RIDServer will run as a service on a designated station directly on a LAN (the Top Producer Hub); it will ensure that all executed copies of Top Producer are legitimate versions of the software.

Consider a customer with 20 computers on a LAN. The customer will receive a separate sheet, providing 20 different RID values. Now each station's MIV will be recorded either at installation time, or the first time Top Producer is executed from that station. All of the RID, MIV, and user-supplied data are maintained by the RIDServer in encrypted form. þ

Further, there is no central file or directory where all of the information is stored. Since the RIDServer will approve program access only for authorized computers (correct RID and correct MIV), the client is assured that as long as he does not give out the software, no one can access his private client information. Note that while the RIDServer must reside directly on the LAN, any station connected to the LAN, direct or via modem, can be checked for having the proper authorization.

The Baitware Process

The RID as used with Burnin allow for the Baitware method of software distribution/anti-piracy. Any means of software distribution can be used for Baitware. This ranges from mass-produced floppies and CD-ROMs to the Internet. The steps of the Baitware method are as follows:

1) Using the Burnin process described above, the application in version $V_{(x)}$ is released as "TrialWare" on date $D_{(x)}$; the application $V_{(x)}$ is BurnIned with the following information:

a) a trial period ($T_{(x)}$) from first execution, after which the program becomes expired. Depending on developer preference, expired programs are disabled via Burnin.

b) an undisclosed "absolute expiry date" or death-date, $D_{(d)}$, set for a date which is given by:

$D_{(d)} = D_{(x)} + T_{(d)}$, where $T_{(d)}$ is a time period of at least 3 times the time period $T_{(u)}$, as described by point 2).

c) a dynamically generated MIV.

d) user information (prompted for).

2) The Burnin process is augmented or enhanced with a new step to re-cover user-registration data from all installations and/or first-executions of the application. This step can be implemented in many forms.

The simplest is to print the registration form (including all user and dynamically generated data) with a mail-back address, then manually enter all data into a database.

The ideal way to implement this step is employ the RIDServer and transmit the registration data to it. The RIDServer maintains a database of all registered customers of the application. In this step, the RIDServer performs the following steps:

1. receive registration data packet,
2. decrypt registration data packet,

3. search database for matching RID,

4. if found

- Store all received information with the "infringer" tag in the database,

- Send back a negative response, indicating that the application should become expired,

5. else (if not found)

- Store all received information in the database; this means the process assumes that the user identified by the input MIV is now the legal owner of the application-copy identified by the input RID.

Note: if implemented as a simple mail-out-form, the steps outlined under the RIDServer would have to be manually performed by the developer.

Within a time period, $T_{(u)}$, after the release date, $D_{(x)}$, a free upgrade of the application in version $V_{(x+1)}$ is released. $T_{(u)}$ is specified by the developer as the time period required for the product to reach 50% of all illegal users. The upgrade $V_{(x+1)}$ will upgrade legal and illegal copies of the application in version $V_{(x)}$. The application $V_{(x+1)}$ will contain patches and responses to user requests. In addition, $V_{(x+1)}$ contains an encrypted list of all registered RID-MIV pairs, as extracted from the database constructed in step 2).

The following steps are performed during the Baitware Upgrade Process to weed out illegal users:

1. search the data base for a matching MIV-RID pair,

2. if found

- reset the absolute expiry date, $D_{(x)}$, to X years from current date; these are the authorized users or paid customers of the application (the reset-date is specified by the developer as the expected lifetime of the application in $V_{(x+1)}$),

3. else (if not found)

- if using a RIDServer, transmit all registration data. RIDServer will simply record all the data for later analysis,
- depending on developer preference, inform user that he/she is an illegal user of the application,
- leave the absolute expiry date, $D_{(x)}$, unchanged,

4. After the absolute expiry date, $D_{(d)}$, the infringer is faced with a situation where

- The installed application $V_{(x+1)}$ no longer works.

Pirating the upgrade $V_{(x+1)}$ does not help either, since the current machine's MIV must match. In fact, the upgrade can perform any action when a non-existing MIV is encountered, including disabling/deleting application data.þ

Baitware not only allows for mass CD-replication and Internet distribution, but also for a nearly full-proof way of preventing piracy.

Concise Restatements of the Invention

Primary Restatement

The primary restatement of the invention (in the most generic and general description) is that the developer can influence installed copies of his application remotely. By remotely, we mean where the developer does not know the physical location and/or ownership of some or all of the installed copies of the application which have been distributed to customers.

Secondary Restatement

A secondary restatement is that the developer can discriminate among those who would try to install the upgrades or updates based on features in his own customer database such as paid/unpaid, age, geographical location, etc. The developer can custom tailor each upgrade with a variety of approaches.

For example :

- Upgrade feature A (say the removal of a drop dead date)- paid users only.
- Up grade feature B (all users, regardless of paid unpaid status)
- UF - C - All valid MIVs of odd number
- UF - D - All valid MIVs of even number
- UF - E - All MIVs of even number
- UF - F - All MIVs of odd number

The choice of odd and even was for purpose of illustration. The developer choose to sort his customer database in some fashion and the upgrades (and hence

his ability to remotely influence each and every copy of the installed base, without having to know where it was located or in whose possession it was stored applies.

It is not the purpose of this method to apply a value judgement on what basis a developer might choose to discriminate among those using his application.

The above description is not intended to limit the claimed invention in any manner, furthermore, the discussed combination of features might not be absolutely necessary for the inventive solution.

The present invention has been described with regard to preferred embodiments. However, it will be obvious to persons skilled in the art that a number of variations and modifications can be made without departing from the scope of the invention as described herein.

THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OF PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1.) The present invention comprises a method and means of preventing software piracy whereby software is treated and distributed according to the Baitware process; the Baitware process comprising the steps of:

a.)Treating the software to use the Burnin process, wherein Burnin performs the initialization steps of:

i.)Calculating a trial period ($T_{(x)}$) from first execution, after which the program becomes expired, whereby expired programs are disabled via Burnin depending on software developer preference,

ii.)Assigning an undisclosed "absolute expiry date" or death-date, $D_{(d)}$, set for a date which is given by:

$D_{(d)} = D_{(x)} + T_{(d)}$, where $T_{(d)}$ is a time period of at least 3 times the time period $T_{(u)}$

iii.)Dynamically generating an MIV value using the current computer software and hardware configurations,

iv.)Prompting the user for user information and the developer supplied RID, where the set of user information is specified by the software developer,

v.)Recording all data from steps i-iv in random and developer-specified locations in the executable file for the software,

vi.)Recording all data from steps i-iv in random and developer-specified locations within the current system-configuration-definition or registry,

b.)Distributing the BurnIned software in version $V_{(x)}$ as TrialWare on the date $D_{(x)}$ via any and all distribution channels, these channels including:

i.)Floppies

ii.) CD-ROM

iii.) Internet

iv.) Intranet

v.)Extranet

c.)Generating and dispensing a new and unique RID value for each copy of software sold to a user, the RID value being unique across all versions of the software throughout its life-time,

d.)Augmenting the Burnin process with a new step to re-cover user-registration data for paid users, including the MIV and RID values, from all installations and/or first-executions of the software, and to further store this data in the Customer Database,

e.)Constructing a free upgrade of the software in version $V_{(x-1)}$ which includes the Customer Database constructed in step d; version $V_{(x+1)}$ being able to upgrade legal and illegal copies of the software in version $V_{(x)}$ via the initialization steps of:

i.)Searching the Customer Database for a matching MIV-RID pair,

ii.)If found, resetting the absolute expiry date, $D_{(x)}$, to Y years from current date, where Y is defined as the life-time of the software in version $V_{(x+1)}$,

iii.)If not found, recording all registration data and dynamically generated data to the central data base of step d, identifying a particular MIV as an illegal, but still upgraded user,

f.)Distributing the software in version $V_{(x+1)}$ constructed in step-e as a free upgrade within a time period, $T_{(u)}$, after the release date, $D_{(x)}$, where $T_{(u)}$ is specified by the software developer as the time period required for the product to reach 100% of currently legal users,

g.)Disabling each copy of the software in version $V_{(x+1)}$ on the death date $D_{(d)}$; this process, which is activated the next time the software is run on or after the death date, comprising the steps of:

i.)Executing and/or undertaking all additional actions specified by the software developer,

ii.)Further disabling the software in version $V_{(x)}$, thereby also disabling any future re-installations of the software in version $V_{(x)}$,

h.)Contacting all illegal users recorded in the Customer Database on the death date $D_{(d)}$, and communicating all software developer specified information.

2. A method and means of software distribution and re-distribution whereby software piracy is eliminated; the process comprising the same steps as 1 above.

3. A method and means of preventing software piracy wherein the released software is protected from infringement by forcing infringers to have to register the software product for continued use.

4. A method and means of preventing software piracy wherein infringers of released software are coerced into purchasing the product for continued use.

5. A method and means of preventing software piracy as in 1 above, wherein each dynamically generated MIV value (1.x) is further associated with user information consisting of the following data:

a.)Name,

b.)Email,

c.)Address,

d.)Phone,

e.)etc.

6. A method and means of preventing software piracy as in 5 above, where the released and branded software (1.x) is further used to distinguish paying users from infringers, such that an infringer's copy of the software also identifies the paying user who illegally re-distributed the software.

7. A method and means of preventing software piracy wherein new market share for the software is forcibly created from the illegal user market for that software.

8. A method and means of preventing software piracy wherein new marketing and distribution channels are forcibly created from the illegal distribution channels for that software.

9. A method and means of preventing software piracy as in 1 above, wherein Copying the installed application from one legal machine to an infringer (even if the registry information is correctly updated) will not enable the infringer to run the application, since the program will generate a different MIV from that which has been burned into its own executable.

10. A method and means of enforcing the terms of any software license by controlling any copy of software after it has left the actual possession of the developer.

11. A method and means of creating a database for maintaining ongoing customer relations.